Comp 6231: Distributed System Design
Winter 2022

Assignment 1: Distributed Appointment Management System (DAMS) Using Java RMI(Remote Method Invocation)

Course Instructor: Prof. R. Jayakumar
Lab Instructor: Mr. Brijesh Lakkad

Student: Mr. Munj Bhavesh Nayak
Student ID: 40195590

Table of Contents

Overview

The Distributed Appointment Management System (DAMS) consists of 3 different hospitals who have their own individual server, totalling 3 servers:

- Montreal Server (MTL)
- Sherbrooke Server (SHE)
- Quebec Server (QUE)

This system consists of 2 types of clients:

- Admins: who carry out the main management of the entire system.
- Patients: the end users of the system.

In this assignment, it has to be ensured that these clients are connected to their own servers with the Java RMI, and also that the connection between the 3 servers happen using the UDP/IP Socket Programming.

Admin Specific Functions:

- addAppointment(): Admins can add 3 different types of appointments (Physician, Dental, and Surgeon) in their own servers.
- removeAppointment(): Admins can remove any previously added Appointments from their respective servers. If any patient had booked an appointment before the admin removed it, the patient must be booked for the closest next available appointment.
- listAppointmentAvailability(): Display all the available appointments of the selected type from all the servers.

Admin/Patient Functions:

- bookAppointment(): Patients can book an appointment from the 3 appointment types and select the date that the admin has added. They can also book an appointment in servers other than their own but it has a weekly limit of 3 appointments.
- getAppointmentSchedule(): show the appointment schedule of the patient.
- cancelAppointment(): patients can cancel their appointments.

The patients are recognized with their patientID consisting of a serverID, clientType (Admin or Patient), and a 4 digit unique identifier key.

Appointments are recognized with their appointmentType: Physician/Surgeon/Dental and appointmentID (serverID + appointmentSlot (M/A/E) + appointmentDate)

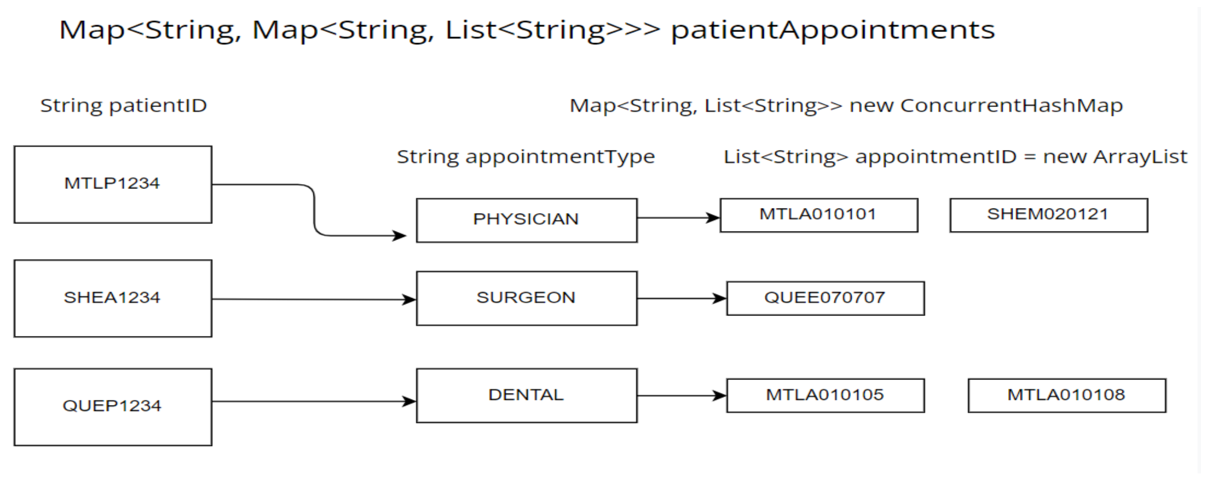
Bothe servers and clients maintain individual log files.

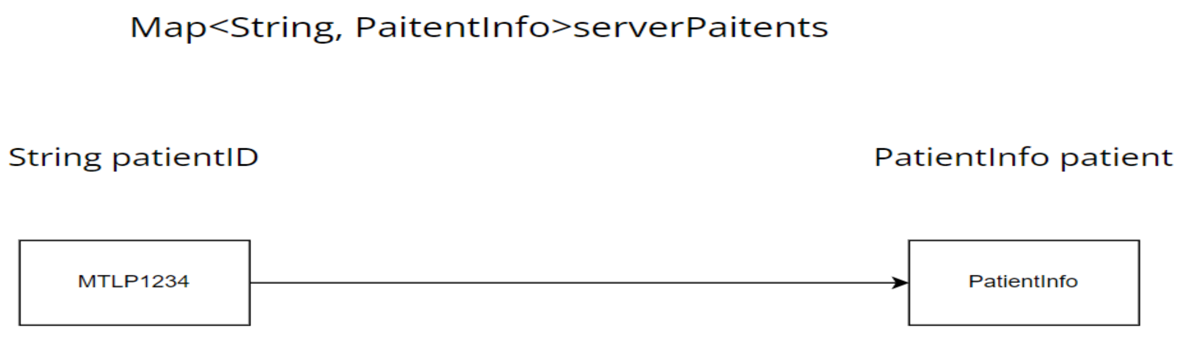
Implementation

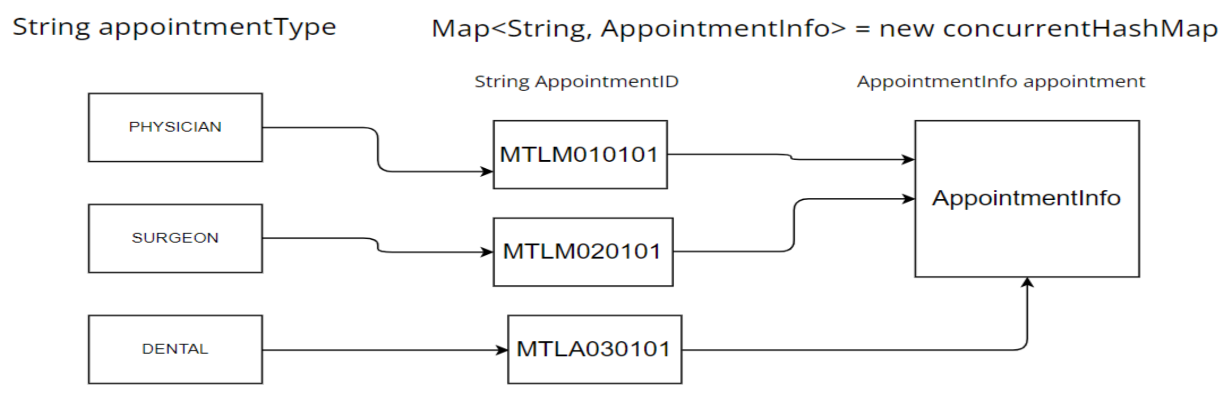
- Client-Server communication is done over RMI
  - Montreal RMI Registry port: 2964
  - Quebec RMI Registry port: 2966
  - Sherbrooke RMI Registry port: 2965
- Server-Server communication is done using the UDP/IP Socket Programming
  - Montreal UDP Port: 8888
  - Quebec UDP Port: 7777
  - Sherbrooke UDP Port: 6666
- To reduce the duplication of the code and ease the development, only one server Implementation is used with 3 instances, as discussed/advised in the lab.
- Clients and Servers maintain separate individual log files.
- Server Files are located at src/Logs/Server/ServerName.txt
- Client Files are located at src/Logs/Client/ClientID.txt
- To ensure maximum concurrency, concurrentHashMaps were used to store the data.
- The most crucial thing in the implementation was to avoid a deadlock/infinite looping in the UDP calls.
- The most challenging part to implement was the AppointmentManager since it is the heart of the entire system dealing with all the important operations.

Data Structures

Following are the DataFlow Diagrams of the Data Structures involved:


Map<String, Map<String, List<String>>> patientAppointments
String patientID
Map<String, List<String>> new ConcurrentHashMap
String appointmentType
List<String> appointmentID = new ArrayList
MTLP1234
SHEA1234
QUEP1234
PHYSICIAN
SURGEON
DENTAL
MTLA010101
SHEM020121
QUEE070707
MTLA010105
MTLA010108





Testing Scenarios

1. Type of Test: Login
   Scenario: UserID
   Cases: AdminID, PatientID
2. Type of Test: Menu
   Scenario: UserID, Logout
   Cases: AdminID, PatientID, Logout
3. Type of Test: Add Appointment
   Scenario: Admin adds an Appointment
   Cases:
   - Invalid AppointmentID: not Added
   - New AppointmentID: Added
   - Existing AppointmentID: Capacity Updated
   - Duplicate Appointment: Not Allowed
   - AppointmentID of other Servers: Not Allowed
4. Type of Test: Remove Appointment
   Scenario: Admin Removes an Appointment
   Cases:
   - Invalid AppointmentID
   - AppointmentID Doesn't Exist
   - AppointmentID with no registrations: Appointment Removed
   - AppointmentID with a registration: get the patient the earliest next available appointment.
   - AppointmentID of other Server: not allowed

5. Type of Test: List Appointment Availability

   Scenario: Listing all available appointments.

   Cases:

   - List all the appointments available from all the servers.
   - Enforce an appointment Type so only that type of appointments are listed.

6. Type of Test: Book Appointment

   Scenario: Patient tries to book an appointment

   Cases:

   - On Own Server: Allowed
   - If capacity is full: not allowed
   - On other servers: weekly limit of 3 appointments.
   - Invalid AppointmentID: not allowed
   - Invalid PatientID: not allowed

7. Type of Test: Get Appointment Schedule

   Scenario: Patient tries to get a schedule of their appointments.

   Cases:

   - Show the appointment Schedule of a patient
   - Invalid patientID: not allowed
   - No bookings: empty return
   - Patient doesn't exist: create new patient

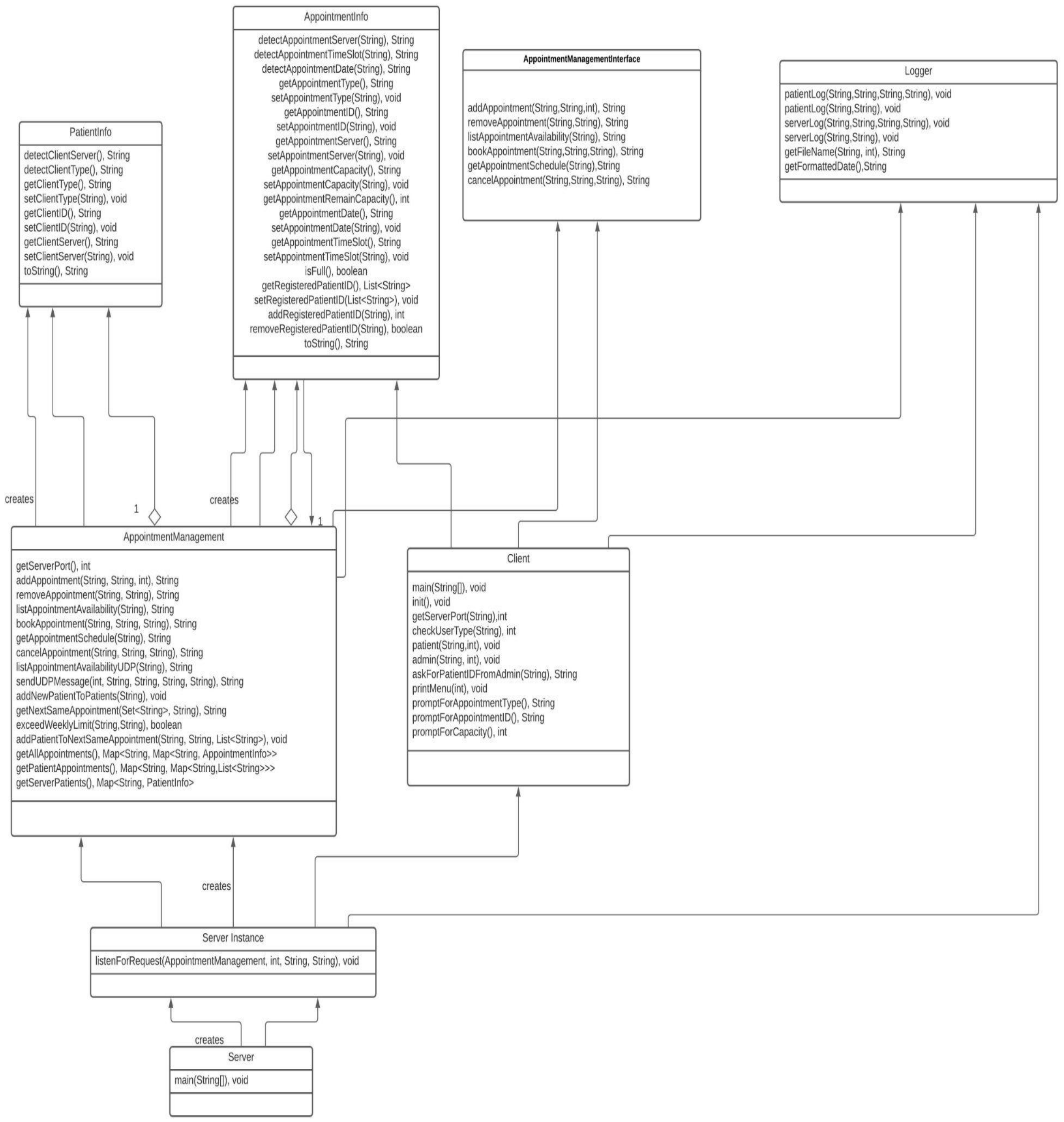
8. Type of Test: Cancel Appointment

   Scenario: Patient tries to cancel an appointment.

   Cases:

   - Cancel at own server: OK
   - Cancel at other Server: OK
   - Cancel a non booked appointment: Not allowed
   - Invalid AppointmentID/PatientID: Not allowed

Class Diagram


AppointmentInfo
detectAppointmentServer(String), String
detectAppointmentTimeSlot(String), String
detectAppointmentDate(String), String
getAppointmentType(), String
setAppointmentType(String), void
getAppointmentID(), String
setAppointmentID(String), void
getAppointmentServer(), String
setAppointmentServer(String), void
getAppointmentCapacity(), String
setAppointmentCapacity(String), void
getAppointmentRemainCapacity(), int
getAppointmentDate(), String
setAppointmentDate(String), void
getAppointmentTimeSlot(), String
setAppointmentTimeSlot(String), void
isFull(), boolean
getRegisteredPatientID(), List<String>
setRegisteredPatientID(List<String>), void
addRegisteredPatientID(String), int
removeRegisteredPatientID(String), boolean
toString(), String
AppointmentManagementInterface
addAppointment(String,String,int), String
removeAppointment(String,String), String
listAppointmentAvailability(String), String
bookAppointment(String,String,String), String
getAppointmentSchedule(String),String
cancelAppointment(String,String,String), String
Logger
patientLog(String,String,String,String), void
patientLog(String,String), void
serverLog(String,String,String,String), void
serverLog(String,String), void
getFileName(String, int), String
getFormattedDate(),String
PatientInfo
detectClientServer(), String
detectClientType(), String
getClientType(), String
setClientType(String), void
getClientID(), String
setClientID(String), void
getClientServer(), String
setClientServer(String), void
toString(), String
creates
1
creates
1
AppointmentManagement
getServerPort(), int
addAppointment(String, String, int), String
removeAppointment(String, String), String
listAppointmentAvailability(String), String
bookAppointment(String, String, String), String
getAppointmentSchedule(String), String
cancelAppointment(String, String, String), String
listAppointmentAvailabilityUDP(String), String
sendUDPMessage(int, String, String, String, String), String
addNewPatientToPatients(String), void
getNextSameAppointment(Set<String>, String), String
exceedWeeklyLimit(String,String), boolean
addPatientToNextSameAppointment(String, String, List<String>), void
getAllAppointments(), Map<String, Map<String, AppointmentInfo>>
getPatientAppointments(), Map<String, Map<String,List<String>>>
getServerPatients(), Map<String, PatientInfo>
Client
main(String[]), void
init(), void
getServerPort(String),int
checkUserType(String), int
patient(String,int), void
admin(String, int), void
askForPatientIDFromAdmin(String), String
printMenu(int), void
promptForAppointmentType(), String
promptForAppointmentID(), String
promptForCapacity(), int
creates
Server Instance
listenForRequest(AppointmentManagement, int, String, String), void
creates
Server
main(String[]), void